Recurrent Natural Language Parsing[1]

Stan C. Kwasny
Department of Computer Science
Washington University
St. Louis, Missouri 63130-4899
sck@cs.wustl.edu

Sahnny Johnson
Department of Mathematics
& Computer Science
Knox College
Galesburg, Illinois 61401-4999
johnson@cs.wustl.edu

Barry L. Kalman
Department of Computer Science
Washington University
St. Louis, Missouri 63130-4899
barry@cs.wustl.edu

Abstract

A recurrent network was trained from sentence examples to construct symbolic parses of sentence forms. Hundreds of sentences, representing significant syntactic complexity, were formulated and then divided into training and testing sets to evaluate the ability of a recurrent network to learn their structure. The network is shown to generalize well over test sentences and the errors that do remain are found to be of a single type and related to human limitations of sentence processing.

Introduction

Problems with time-dependent input require computational mechanisms that can respond well over time. One such area, natural language processing, is seen by some as pivotal to the success or failure of connectionism. Pinker and Prince (1988, p.78) stated that "Connectionism, as a radical restructuring of cognitive theory, will stand or fall depending on its ability to account for human language." Meeting this challenge is a major focus of our work.

From the basic design of a deterministic natural language parser, we are introducing and evaluating connectionist techniques and evolving toward a fully connectionist language understanding system. Deterministic (wait-and-see) parsing (Marcus, 1980) is a good model for the design of a neural network based parser fore two reasons. First, it concentrates on syntax, which has the advantage of being relatively well understood and therefore leads to results which can be easily compared. Second, the fixed-length input buffer of a deterministic parser is consistent with the input strategy of recurrent neural networks, such as those due to Elman (1990), which iteratively process fixed-sized pieces of a larger input. The result is a natural fit between a deterministic approach to natural language processing and the processing strategy of a recurrent net.

Motivation

Marcus (1980, p. 204), in his determinism hypothesis, conjectured that "there is enough information in the structure of natural language ... to allow left-to-right deterministic parsing of those sentences which a native speaker can analyze without conscious effort." A clear implication of this hypothesis is that natural language processing need not depend in any fundamental way on backtracking strategies. Furthermore, parsing should not be a process which generates partial structures from false starts that are eventually discarded in favor of others. While this hypothesis does not dictate any details about the architecture or the rules, it does limit sentences to those that a native speaker can analyze without conscious effort. In effect, this eliminates garden-path sentences such as "The horse raced past the barn fell."

Marcus constructed PARSIFAL, a system that examined this hypothesis under a specific rule-based architecture involving a stack, a set of often complicated rules, rule packets, rule priorities, and an intricate mechanism of "attention-shifting rules. The true force of the hypothesis, as it pertains to human language processing, was left unexamined.

We are testing the hypothesis directly by developing a corpus of sentences which a native speaker can analyze without conscious effort, and constructing a neural net architecture that learns how to process sentences from examples. The trained network accepts a buffer of sentence elements presented in sequence and produces structure building and manipulating (primitive) actions as output. Such a performance-oriented approach (as opposed to a competence-oriented one) is intentional. There is growing belief that "... human parsing resources must be characterized as finite-state" (Pulman, 1986) and that full recursion, as supported by context-free languages, may be unnecessary (Christiansen, 1992; Church, 1982). Simple recurrent networks possess the power of a finite state machine, but as such can approximate, to a finite degree, the processing requirements of a context free language.

Unlike recurrent parsing networks described elsewhere (see, for example, Das et al., 1992), our system contains no explicit internal or external stack. The stack's role is assumed to a sufficient degree by the recurrence in the network.

Weckerly and Elman (1992), in a study similar to ours, trained a recurrent network with center-embedded sentences to examine simple semantic preferences. They found that a stack-like approach, one in which levels of processing are tightly encapsulated, is incorrect if one is to account for human processing of semantically biased verbs. The study described here is more ambitious in its scope, but is limited at the moment to syntactic processing only.

The test of whether there is enough information in the structure of natural language to allow deterministic parsing, as the deterministic hypothesis claims, becomes an evaluation of the extent to which training enables the recurrent network to generalize to novel sentence forms. Our approach involves exposing the network to a variety of complex sentences during training and then testing for generalization on a completely different set of sentences. Training and testing data for the network is developed by tracing the symbolic processing of

[1]This material is based upon work supported by the National Science Foundation under Grant No. IRI-9201987.

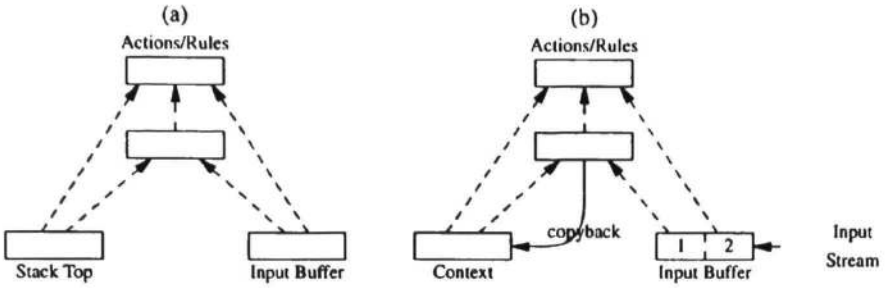

Figure 1: Network architectures. Dotted arrows represent full connectivity between layers.

a corpus of (non-garden-path) sentences. A total of 209 sentences, each unique in the processing steps required, are used in this experiment. These are randomly divided into a set of 156 training sentences and a set of 53 testing sentences. Generalization is shown to occur and the errors in processing provide useful insights into the nature of this process.

System Architecture

While our design departs substantially from PARSIFAL, we follow many of the same ideas. Parsing is performed deterministically in that system by permitting the parser to look ahead up to the next three constituents of the sentence, which are held in a three-place buffer. PARSIFAL requires a stack to allow the recursive processing of embedded structures and to facilitate processing generally. Primitive actions which build structure and move constituents can be performed on the stack and in the buffer positions. These actions are controlled by a set of grammar rules. The rules are usually associated with the current (top-level) node of the structure being built, which is held on the top of the stack. A processing step consists of selecting an applicable rule and firing the rule by performing its action, forcing changes to the stack and/or buffer. After a series of steps, a termination rule fires which ends the processing and the final structure is left on top of the stack.

Previous Work

In earlier work, we replaced the rules used by PARSIFAL with a feedforward neural network trained from either rules or sentences. Our goals in that work were to unify PARSIFAL and a variety of proposed extensions and amendments within one model and to test the ability of the parser to address issues of ambiguity and ill-formedness. The network learned a mapping from the patterns used to encode the sentence elements in the buffer and the structure on top of the stack to the appropriate rule and its action. The three-place buffer, stack, and actions were all handled symbolically. Figure 1(a) illustrates the network design used in our earlier work (Kwasny & Faisal, 1990; Kwasny & Kalman, 1991; Kwasny & Faisal, 1992). Casting the rules as a neural network yielded significant advantages in robustness and we have studied lexical ambiguity, grammatical ill-formedness, lexical omission, and other properties of the design.

Current Work

In our current work, a simple recurrent network (Elman, 1990) is utilized which permits some degree of memory of past decisions. Although stack-like processing is necessary to address embeddings of the type present in all context-free languages, the recurrent design, shown in Figure 1(b), obviates the need to present the top-of-stack contents to the network as in our previous work because sufficient information is represented in the recurrent connections. Because the choice of appropriate rule in a symbolic deterministic grammar is largely dependent on information stored on the stack, heavy responsibility in our adaptive network falls on the recurrent connections. The network is coerced, through training, to produce the primitive *create* and *drop* actions that manipulate the stack without explicitly representing the stack. The output is a sequence of actions to be performed on the symbolic structures that evolve during processing.

During processing, the activation pattern of the hidden layer in the recurrent network is copied back to the input layer on each step. These activation patterns encode information about relevant past events so that current and future decisions can be influenced, much as the stack would influence symbolic processing. Since there is evidence that isolating the linearly separable relationship from the non-linear part makes for more effective training (Lee and Holt, 1992), and since parsing seems to require approximately linear processing for most of the steps, we adapted the recurrent network designed by Elman to include direct connections from input to output units. The parser is illustrated in Figure 2.

With this design, we are moving closer to a confrontation with the determinism hypothesis: Can a parser extract the structure of English sentences left-to-right deterministically as the hypothesis claims? The real evidence is in the complexity of sentences we have processed, in the extent of coverage of the structures of English, and in the degree of success achieved.

Training

Patterns must be obtained that illustrate, by example, the mapping to be learned. This is true whether we are training a recurrent network or training a feedforward network. The primary difference comes in the sequencing required for recurrent networks. Patterns must be arranged in the sequence

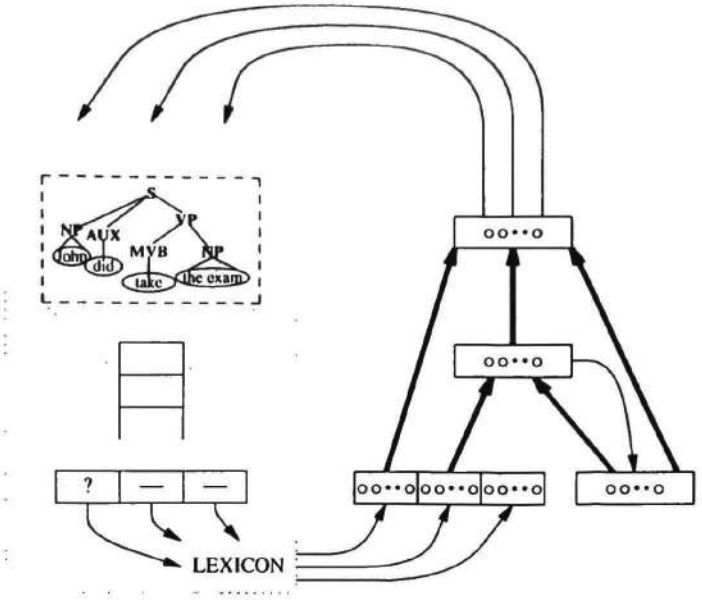

Figure 2: Parser design. The buffer contents are transformed using the lexicon and presented to the recurrent network which produces an action that is performed on the symbolic structures.

in which they occur during processing. This is necessary to permit proper context patterns to be computed and derived as a by-product of training. The word order of the sentence forms used for training provides the sequencing.

Figure 3 shows a sample of the 156 randomly selected training sentences. Each sentence is processed symbolically from left to right in a manner analogous to that of PARSIFAL. A lexicon is used to determine the appropriate pattern of syntactic features for each word. Finch and Chater (1992) argue that such syntactic features can be learned from the statistical properties of sentences in a domain, although our lexicon was built manually, in the interest of time, to fit our examples.

For the particular set of rules we are using, only two buffer positions are required, but the technique can easily be extended to a longer buffer if required. (Marcus argues that three is sufficient for English.) The first buffer position is encoded using 26 units while the second buffer position requires 12 units. These units represent features such as part of speech (noun, verb, auxiliary, etc.), verb form (infinitive, gerund, past participle, etc.), noun phrase, verb phrase, final punctuation, and others.

Example

If the first two words of the sentence were 'the cat', for example, two units in the representation of the first buffer position would be turned on to indicate that the first word is a determiner and that it is a symbol that can begin a noun phrase. The second buffer pattern would indicate that the second word is a singular noun. The symbolic parser also needs to examine the topmost stack node since rules depend on the constituent being built. The top of the stack would be empty in this example. So, the parser selects the rule which is applicable when there is a determiner in the first buffer position, a singular noun in the second position, and the stack is empty. The action executed will be to create (push) a new sentence (S) node on the stack. The patterns from the buffer (but not the stack) and the pattern representing the selected output are collected at each step. In this way, we end up with a sentence trace in the form of a sequence of coded condition-action patterns suitable for training. The buffer patterns provide the input stream and the actions chosen at each step are the desired output.

While the patterns could be derived by hand, this would not be expeditious. In practice we use a set of grammar rules designed for this purpose. They are applied automatically to assure that consistent rule application and coding practices are followed. The grammar is based on one used in PARSIFAL and consists of 76 single-action rules, each of which selects one of 39 possible actions. Among the actions that may be performed are: attach as an adjective, auxiliary, main verb, etc; create a noun phrase node, verb phrase node, secondary sentence node, etc; switch (interchange contents of first two buffer positions); drop (pop the stack, shifting the top of the stack into the first buffer position); and stop. For the examples presented in this paper, 24 hidden units are required, which are copied back at each recurrent step.

Reducing Training Complexity

Training is time consuming, but possible. Use of singular value decomposition (SVD) to transform the input patterns into a space in which the input units are aligned orthogonally is one primary reason for success in training and seems to be particularly important for sizable networks with large numbers of training patterns See Kalman et al. (1993) for a complete discussion of this technique and results obtained. This method leads to quicker and more effective training, and permits the resulting weights to be back-transformed so that they perform identically on the original training patterns. SVD allows easy identification of those inputs that contribute weakly or not at all to the problem. For the parser, the number of input units was reduced from 38 to 34, effectively yielding an architecture

1 John likes Mary.
8 The tall man scheduled the meeting.
17 The angry old man has punished the little boy.
27 Books were read by the boys.
32 Was John taken in the house?
38 The big angry old man would like the little kitten for the barn.
51 Will he have gone by Friday?
69 Does Mary like the cat?
86 Mary has been punished by the teacher who scheduled the exam for Friday.
104 Was it scheduled by the teacher who complained?
120 Has the man bought the big books which the tall boys read?
132 Was the black cat taken by the girl who liked it?
138 The books which were read by Mary have been taken to the meeting which John scheduled.
141 Take the books which the man read from the house which the teacher likes!
144 The tall man who likes Mary scheduled the meeting which the teacher wants for the boys who should have been punished.
151 The man who the girl who disappeared likes scheduled the meeting.
154 The boy who likes the girl who likes the cat bought the book.
156 They bought the books which the teacher who scheduled the exam which John read liked.

Figure 3: Sample of Training Sentences

2 John scheduled the meeting for her.
5 Mary has scheduled a meeting for Wednesday.
9 Do boys schedule meetings?
12 John was punished by him.
16 He could have been punished by the tall man.
18 Should Mary have been taken to the meeting?
22 John scheduled a big meeting.
25 She likes the cat.
28 She has met her.
31 The books which the teacher bought were read by the boys.
33 Mary likes the boy who likes the cat.
35 John was punished by the teacher in the barn which would fall on Friday.
41 Boys who like the teacher disappeared.
42 The boys who could have bought the horse want the teacher in the house on Friday.
45 Should the girl have been taken to the house which fell?
47 The man who the girl likes has punished the boy who likes Mary.
49 Does she like the old books which the teacher scheduled for the exam which disappeared?
50 Were the books which the teacher had bought read by the man who prays in the barn which would fall on Friday.
53 John scheduled a meeting for the boy who would take the exam which the teacher scheduled.

Figure 4: Sample of Testing Sentences

Error Frequency	Targets:				
Network Output:	Drop	Attach PP-NP	Attach Obj-NP	Attach Punct	Attach Subj-NP
Drop		0,1	1,1	2,1	0,1
Attach PP-NP			1,5		
Attach Obj-NP		3,0			
Attach Punct	2,6				
Attach Subj-NP	2,0	0,1	1,0		
Create Aux	1,4				

Figure 5: Error Analysis: Frequency of errors occurring in the actions reported by the network vs. targeted actions. Only the non-empty portion of the 39 × 39 table is shown. The first value is from training and second value is from testing.

of 58-24-39 for training and 62-24-39 for testing. Training itself is performed using a modified conjugate-gradient method (Kalman, 1990) that has been implemented to utilize the parallelism available on a 20-processor Sun SPARC Center 2000 machine.

Results

Training proceeded until the number of errors encountered in the training set was small and further training increased the number of errors in the testing set. The training set of 156 sentences were trained until only 13 errors remained out of 5,465 total steps represented in the sentences (0.2% error). The performance of the trained network was then evaluated on the 56 randomly selected testing sentences. A sample of these sentences is shown in Figure 4. Each sentence was processed as a sequence of items which flows into the buffer and was encoded for presentation to the network. The stack was absent from this process and therefore the performance of the parser hinges on the degree to which the recurrent network has generalized the sequential processing.

Collectively, there are 1,860 parsing steps in processing the 53 sentences correctly. Of these, 1,840 steps (99%) were correctly generated by the network and 20 (1%) were not. On the sentence level, this translates into 40 sentences (75%) processed completely without error and 13 sentences (25%) that contained at least one error during processing.

Discussion

This paper has presented evidence that complicated sentence processing can be learned effectively by a recurrent neural network. While the overall performance of the parser is quite good, the errors that do occur are worthy of further discussion.

Analyzing Errors

The errors are not random. None of the errors occur within a constituent. That is, the network apparently has learned the proper sequence of actions relating to the base grammar and its ability to construct lower level pieces of constituents. The errors that do occur cluster around constituent boundaries. Every 'create X' action must be eventually answered by a 'drop' action and a subsequent 'attach X as Y' action. To accomplish this requires, at minimum, a context-free grammar mechanism necessitating the use of a stack. The recurrent network does not contain the equivalent of a stack, but can approximate such a mechanism to a finite extent.[2] For this reason, sentences requiring long-range dependencies of the sort that could be captured by a simple stacking mechanism are occasionally missed.

All of the errors relate to the actions 'drop' and 'attach' mentioned above. Figure 5 illustrates the confusion between attaching and dropping. In one category of error, 'drop' is the target and 'create aux' is the network response. This happens when the constituent boundary for a subject NP modified by an embedded sentence is missed and the parser wants to continue with processing of the main sentence.

In missing a drop action, a typical situation would be a clause at the end of a sentence wherein a final punctuation mark occupies the first buffer position and a secondary sentence is on top of the stack. The network may attempt to attach the final punctuation to the embedded secondary sentence rather than dropping the constituent and attaching both constituent and final punctuation to the main sentence node. In missing an attach action, the network typically substitutes the wrong label. Thus, the parser would substitute for an 'attach pp-np' action, which creates the dominating node structure for the object of a preposition, an 'attach obj-np' action, which identifies it as the object of the sentence. All of these error examples can be seen as related to a similar underlying difficulty, namely that once a node is created, a later step must take action to place it into the structure.

Some of these errors can be overcome with a slightly different scheme for identifying the dominating node in a structure. At creation time, the role of the constituent (e.g., Subj-NP, PP-NP, etc.) is known and therefore the final label can be created along with the node. Later, at the time of the attach, no label need be supplied. This will effectively collapse all attaches into one action.

Other errors are more fundamental. A recurrent network cannot overcome the memory limitations inherent in a finite state device. We hope to eventually show that these errors are comparable to those that humans make during sentence processing.

[2] Any computational mechanism is memory limited and, therefore, cannot fully implement an unbounded stack, although for practical purposes this is usually not a problem. A recurrent network also has limited resources in that feedback only contains a finite number of units. The difference is in the fact that a recurrent network cannot dynamically request more resources as it performs.

Future Work

Several ideas stemming from this work are being investigated.

The granularity of the primitive actions taken from PARSIFAL has contributed to the errors. We are actively revising the set of primitive actions so that frequently occurring combinations will be fused into a single action. For example, drop actions are often followed by attach actions and an action could easily be constructed that would finalize a constituent and attach it properly under its dominating node.

A similar modification involves the dependency between create and attach. Now, a create action for a node and its corresponding attach action must both specify the node label. For example, a 'create PP' action eventually requires an 'attach PP' action to adjoin the PP structure it has built. This could be changed so that the attachment need not redundantly label the constituent.

Work is ongoing to investigate the proper inclusion of semantic information in the parser. This process is requiring major changes in which lexical representation is being integrated with semantic structure building in a micro-world. Whether the finite-state hypothesis continues to prove sufficient is yet to be determined.

Acknowledgements

We are grateful to Morten Christiansen for discussions on this work and to Peter McCann for comments on earlier drafts.

References

Christiansen, M. (1992). The (Non) Necessity of Recursion in Natural Language Processing. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 665–670). Hillsdale, NJ: Lawrence Erlbaum Associates.

Church, K. (1982). On Memory Limitations in Natural Language Processing. Bloomington, IN: Indiana University Linguistics Club.

Das, S., Giles, C. L. & Sun, G.Z. (1992). Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 791–796). Hillsdale, NJ: Lawrence Erlbaum Associates.

J. L. Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179–212.

Finch, S. & Chater, N. (1992). Bootstrapping Syntactic Categories. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 820–825). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kalman, B.L. (1990). Super linear learning in back propagation neural nets (Tech Rep WUCS-90-21). St. Louis: Washington University, Department of Computer Science.

Kalman, B. L., Kwasny, S. C. & Abella, A. (1993). Decomposing input patterns to facilitate training. In *Proceedings of the World Congress on Neural Networks* (pp. III-503–III-506). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kwasny, S. C. & Kalman, B. L. (1990). Connectionism and determinism in a syntactic parser. *Connection Science*, 2(1&2), 63–82.

Kwasny, S. C. & Kalman, B. L. (1991). The case of the unknown word: Imposing syntactic constraints on words missing from the lexicon. In *Proceedings of the Third Midwest Artificial Intelligence and Cognitive Science Society Conference* (pp. 46–50). Carbondale, IL: Midwest Artificial Intelligence and Cognitive Science Society.

Kwasny, S. C. & Faisal, K. A. (1992). Symbolic parsing via sub-symbolic rules. In J. Dinsmore (Ed.), *Closing the Gap: Symbolism vs. Connectionism*. (pp. 209–235). Hillsdale, NJ: Lawrence Erlbaum Associates.

S. E. Lee, S. E. & Holt, B. R. (1992). Regression analysis of spectroscopic process data using a combined architecture of linear and nonlinear neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (pp. IV-549–IV-554). Piscataway, NJ: IEEE.

Marcus, M. P. (1980). *A theory of syntactic recognition for natural language*. Cambridge, MA: The MIT Press.

Pinker, S. & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (Eds), *Connections and symbols*. (pp. 73–193). Cambridge, MA: The MIT Press.

Pulman, S. G. (1986). Grammars, parsers, and memory limitations. *Language and Cognitive Processes* 2, 197–225.

Weckerly, J. & Elman, J. L. (1992). A PDP approach to processing center-embedded sentences. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 414–419). Hillsdale, NJ: Lawrence Erlbaum Associates.